Tampereen yliopisto

Markus Halme

GENEETTINEN ALGORITMI JA SEN SOVELLUKSET

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Toukokuu 2021

TIIVISTELMÄ

Markus Halme: Geneettinen algoritmi ja sen sovellukset
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2021

Geneettinen algoritmi on evoluutioalgoritmi, joka käyttää hyväksi evoluutiobiologiasta tuttuja prosesseja. Geneettistä algoritmia käytetään usein optimointimenetelmänä joko suoraan ongelmien ratkaisemiseksi tai toisten algoritmien tukena. Geneettisen algoritmin pitkän historian takia siihen liittyvää kirjallisuutta on paljon, ja sitä on mahdollista tarkastella monesta eri näkökulmasta.

Tässä tutkielmassa syvennyn geneettiseen algoritmiin, sekä sen sovellusalueisiin kahden nykypäivän sovelluksiin hyvin oleellisesti liittyvän ongelman kautta. Geneettisen algoritmin rakenne muovautuu hyvin paljon käsillä olevan ongelman mukaan, jonka takia geneettistä algoritmia esitellessäni käytän yksinkertaisen geneettisen algoritmin (simple GA, SGA) rakennetta. Työn tavoite on pyrkiä vastaamaan nykypäivän kirjallisuuden pohjalta, että millaisten ongelmien kanssa geneettistä algoritmia voidaan hyödyntää ja kuinka se käytännössä tapahtuu. Tutkielma on kirjallisuuskatsaus geneettiseen algoritmiin ja sen sovelluksiin ja aineisto koostuu suurimmaksi osaksi tuoreista artikkeleista. Vanhempia artikkeleita käytän pääosin aiheen taustoittamiseen ja käsitteiden määrittelyyn.

Tutkielmassa tarkastellaan geneettisen algoritmin soveltamista kauppamatkustajan ongelmaan sekä reppuongelmaan. Kauppamatkustajan ongelmassa keskeiseksi nousi oikean risteytysmenetelmän määrittely, minkä takia tutkielmassa käydään läpi erilaisia tapoja suorittaa risteytys. Reppuongelmassa sovelletaan yksinkertaista geneettistä algoritmia, sekä syvennytään nykypäivän kirjallisuuden avulla erilaisiin tapoihin muokata geneettistä algoritmia kyseisen ongelman ratkaisemisen optimoimiseksi.

Avainsanat: geneettinen algoritmi, kauppamatkustajan ongelma, reppuongelma, sovelluksia

Sisällysluettelo

1 Johdanto

Geneettinen algoritmi on yksi tunnetuimmista evoluutioalgoritmeista (Eiben & Smith, 2015). Geneettisen algoritmin käsitettä käytti ensimmäistä kertaa John Holland kirjassaan *Adaptation in Natural and Aritifial Systems* vuonna 1975. Evoluutio-ohjelmointi kuitenkin yleistyi vasta 1980-luvulla, kun tarpeeksi laskentatehoa oli saatavilla (Reeves, 2010). Geneettinen algoritmi pystyy luonteensa takia käyttämään hyödyksi rinnakkaisuutta (Man et al., 1999) ja täten tarjoamaan mahdollisuuden ratkaista ongelmia, joihin monet muut algoritmit eivät pysty. Geneettiselle algoritmille on myöhemmin löydetty sovellusalueita niin tieteelliseltä kuin teknilliseltäkin puolelta (Man et al., 1999).

Tutkielmani tavoitteena on selvittää mihin ja miten geneettistä algoritmia käytetään. Tarkastelen myös geneettisen algoritmin käyttöä kahden abstraktin ongelman ratkaisemiseksi, joita voidaan soveltaa nykypäivän käytännön ongelmiin. Tällaisia nykypäivän ongelmia ovat esimerkiksi tietoverkon reititys tai ohjelmiston resurssienhallinta. Geneettistä algoritmia hyödynnetään nykyään usein optimointiongelmiin, vaikkakin Hollandin alkuperäisessä työssä optimoinnin rooli oli hyvin pieni (Reeves, 2010).

Tutkielmani on kirjallisuuskatsaus geneettiseen algoritmiin ja sen sovelluksiin. Tarkoituksena on luoda lukijalle ymmärrys geneettisen algoritmin käytöstä ja sen soveltamisesta nykypäivän ongelmiin nojautuen niin vanhempaan kuin uudempaankin kirjallisuuteen. Vanhemman kirjallisuuden ideana on luoda pohjaa tutkielmassa käytetyille käsitteille ja perinteisille metodeille, joita geneettisen algoritmin kanssa käytetään. Uudempi kirjallisuus taas tuo näkökulmia siihen, kuinka näitä perinteisiä metodeja on paranneltu ja täydennetty.

Tässä työssä käyn aluksi luvussa 2 läpi geneettistä algoritmia ja sen pääpiirteitä yksinkertaisen mallin kautta. Tämän jälkeen luvussa 3 esittelen ja syvennyn kahteen eri ongelmaan – kauppamatkustajan ongelmaan ja reppuongelmaan – joita tarkastelen geneettisen algoritmin näkökulmasta. Nämä ongelmat ovat valittu siksi, että ne tarjoavat abstraktin tavan tarkastella geneettisen algoritmin soveltamista, jolloin itse ongelman määrittely on yksinkertaisempaa eikä tutkielma paisu liikaa. Lopuksi luvussa 4 teen yhteenvedon tutkielmassa käsitellyistä asioista.

2 Geneettinen algoritmi

Geneettisen algoritmin tarkoituksena on pyrkiä löytämään paras mahdollinen ratkaisu käsillä olevaan ongelmaan järjellisessä ajassa käyttäen hyväksi evoluutiobiologian keskeisiä käsitteitä kuten *luonnonvalintaa* (natural selection) (Man et al., 1999). Geneettisellä algoritmilla ei kuitenkaan ole yhtä tiettyä rakennetta vaan se muovautuu käyttötarkoituksen mukaan (Eiben & Smith, 2003). Tässä työssä geneettisen algoritmin esittämisen pääpainona toimii *yksinkertainen geneettinen algoritmi* (simple GA tai SGA), sillä se sisältää geneettisen algoritmin pääpiirteet, sekä tarjoaa nimensä mukaisesti yksinkertaisen tavan aiheen esittämiseen. Tämän takia sitä käytetäänkin usein geneettisen algoritmin opettamiseen (Eiben & Smith, 2015).

Geneettisen algoritmin kanssa on tärkeää tietää muutamia genetiikan käsitteitä. Seuraavaksi esittelemieni käsitteiden tulkinnat tulee pitää vain tämän työn kontekstissa, sillä käsitteillä ei ole geneettisen algoritmin tutkimuksissa yhtä selvää linjausta, vaan käsitteillä saatetaan tarkoittaa jotain muuta muiden töiden yhteydessä.

Eiben ja Smith (2015) määrittelevät *genotyypit* (genotype) ja *fenotyypit* (phenotype) niin, että genotyyppien voidaan kuvitella koodaavan fenotyyppejä. Genotyypit koostuvat taas *geeneistä*. Esimerkiksi voidaan ajatella, että binäärimerkkijonolla tarkoitetaan genotyyppiä, kun taas fenotyypillä sitä lukua, jota binäärimerkkijono esittää. Joissain teoksissa genotyypistä käytetään käsitettä *kromosomi* (chromosome) (Mitchell, 1999), mutta sekaannuksen välttämiseksi käytän tässä työssä vain genotyypin käsitettä. Eibenin ja Smithin (2003) mukaan *lokus* (locus) tarkoittaa genotyypin sisällä olevaa sijaintia ja *alleeli* (allele) sijainnissa olevaa objektia. Geenillä voi olla binäärimerkkijonon tapauksessa siis kaksi eri alleelia, 1 tai 0. Esimerkiksi genotyypin 1010 yksi lokuksista on sen nollaindeksi ja tämän alleeli luku 1.

Koodiesimerkki 1 esittelee yksinkertaisen mallin geneettisestä algoritmista sisältäen pääpiirteet ja idean siitä kuinka geneettinen algoritmi etenee. Jokaista toistorakenteen iteraatiota kutsutaan *sukupolveksi* (generation) (Mitchell, 1999).

```
{
   Alusta populaatio
   WHILE (lopetuksen edellytyksiä ei ole saavutettu) {
      Arvioi populaation yksilöt sopivuusfunktiolla
      Valitse yksilöt risteytykseen
      Risteytä valitut yksilöt (→ uuden sukupolven populaatio)
      Suorita mutaatio uuden populaation yksilöille
      Korvaa vanha populaatio uudella
   }
}
```

Koodiesimerkki 1. Geneettisen algoritmin suoritus.

2.1 Populaation alustus

Populaation tarkoituksena on säilöä mahdollisia ratkaisuja ja toimia täten genotyyppien multijoukkona (Eiben & Smith, 2015). Tärkeää tässä vaiheessa on miettiä kuinka suuri populaation tulisi olla ja miten populaatioon tulisi valita yksilöitä (Reeves, 2010).

Populaation koon määrittäminen oikein on tärkeää, koska sillä on perustavanlaatuinen vaikutus geneettisen algoritmin tehokkuuteen ja toimivuuteen (Koumousis & Katsaras, 2006). Liian pieni populaation koko voi aiheuttaa konvergoitumisen epäoptimaaliseen ratkaisuun, kun taas liian suuri populaation koko lisää vaadittua laskentatehoa (Koumousis & Katsaras, 2006), joka vaikuttaa algoritmin suorittamiseen tarvittavaan aikaan.

Yksi tapa valita populaation yksilöt on satunnaisvalinta (Man et al., 1999). Reevesin (2010) mukaan, koko populaatiota tarkastellessa pitäisi olla mahdollisuus löytää jokainen alleeli jokaisesta genotyypin lokuksesta. Esimerkiksi kahden merkin binäärimerkkijonojen tapauksessa populaatio, joka sisältää genotyypit 10 ja 01 tyydyttää tämän ehdon, koska kummastakin lokuksesta voidaan löytää alleelit 1 ja 0. Reeves (2010) ehdottaakin populaation koon löytämistä etsimällä pienin populaation koko, joka on samalla tarpeeksi suuri antamaan merkityksellisen tuloksen hyvin todennäköisesti. Esimerkiksi binäärimerkkijonojen tapauksessa (olkoon l merkkijonon pituus ja M valittu populaation koko)

$$P = (1 - (1/2)^{M-1})^l$$

Kaava 1. Binäärimerkkijonopopulaation koko.

nähdään että populaatio, jonka koko on 17, ja jonka sisältämien merkkijonojen pituus on 50, todennäköisyys jokaisen alleelin löytämiseksi jokaisesta merkkijonon lokuksesta populaatiossa on yli 99,9 % (Reeves, 2010). Näin on mahdollista minimoida todennäköisyys, että parhaimpaan ratkaisuun tarvittava alleeli puuttuisi alkuperäisestä populaatiosta.

Satunnaisvalintaa on myös mahdollista muokata sisällyttämällä luotuun populaatioon jo aiemmin hyviksi todettuja ratkaisuja, jolloin parhaan ratkaisun löytämistä on mahdollista nopeuttaa (Reeves, 2010). Näiden jo aiemmin hyväksi todettujen ratkaisujen löytämiseen on mahdollista käyttää useita eri heuristisia metodeja, joihin tässä työssä ei kuitenkaan syvennytä muuten kuin mainitsemalla niiden olemassaolo.

Alustetaan populaatio, jolla on mahdollista löytää toisen asteen yhtälön $-1x^2 + 6x$ maksimoiva luku x väliltä 0–15. Valitaan populaation kooksi 13, sillä populaatio luodaan 99,9 % todennäköisyydellä sellaiseksi, että jokainen alleeli on edustettuna jokaisessa lokuksessa, kun tarkastellaan koko populaatiota. Ratkaisemalla populaation koko eli M kaavasta 1 (parametrein P = 0.999 ja l = 4), saadaan luku 13. Kaava 2 havainnollistaa tilannetta, jossa kaava 1 on muokattu sellaiseksi, että siitä voidaan ratkaista populaation koko.

$$1 + \frac{\ln\left(1 - \sqrt[4]{0.999}\right)}{\ln\left(0.5\right)} \approx 13$$

Kaava 2. Populaation koon ratkaisu kaavasta 1.

Koodiesimerkki 2 havainnollistaa populaation luomista.

```
{
    WHILE (populaation koko < 13) {
        generoi 4 eri satunnaislukua väliltä 0-1 ja pyöristä luvut lähimpään
        kokonaislukuun
        lisää luvut uuteen yksilöön ja säilö tämä yksilö populaatioon
    }
}
```

Koodiesimerkki 2. Neljän merkin binäärimerkkijonopopulaation luonti.

2.2 Sopivuusfunktio

Eibenin ja Smithin (2015) määritelmän mukaan sopivuusfunktio pyrkii ohjaamaan populaation muutosta oikeaan suuntaan ja määrittelemään mitä populaation edistyminen oikein tarkoittaa. Tämän funktion perusteella on mahdollista selvittää mitkä yksilöt edusta-

vat parempia ratkaisuja ja antaa näille yksilöille paremmat mahdollisuudet valikoitua risteytykseen (Reeves, 2010).

Tyypillisesti sopivuusfunktion tehtävä on kääntää genotyypit fenotyypeiksi ja tämän jälkeen arvioida fenotyyppien laatua (Eiben ja Smith, 2015). Esimerkiksi tilanteessa, jossa tavoitteena on löytää toisen asteen yhtälön maksimoiva luku x, sopivuusfunktio kääntää aluksi binäärimerkkijonot luvuiksi (0010 →2) ja sen jälkeen sijoittaa luvut toisen asteen yhtälöön ja säilöö tulokset. Näiden tulosten pohjalta on mahdollista vertailla jokaista populaation yksilöä ja selvittää mitkä yksilöt ovat parempia kuin toiset.

```
{
    WHILE (populaation kaikki alkiot läpikäymättä) {
        muunna genotyyppi fenotyypiksi (0010 →2)
        aseta muunnettu luku toisen asteen yhtälön x:n paikalle
        säilö yhtälöstä saatu luku
    }
}
```

Koodiesimerkki 3. Sopivuusfunktio toisen asteen yhtälön maksimoivalle x:lle.

2.3 Valinta

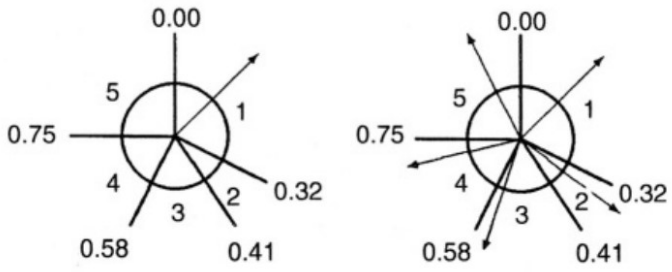
Kun yksilöt on ajettu sopivuusfunktion läpi, on seuraavaksi päätettävä miten yksilöitä valitaan seuraavaan vaiheeseen eli risteytykseen. Tämän valinnan voi suorittaa usealla eri tavalla. Risteytyksen tuloksena syntyneitä uusia yksilöitä kutsutaan *lapsiksi* (child) ja risteytettyjä yksilöitä *vanhemmiksi* (parent).

Eibenin ja Smithin (2015) mukaan yksinkertaisin tapa suorittaa valinta on *rulettivalinta* (roulette-wheel selection). Rulettivalinnassa yksilöille annetaan mahdollisuus valikoitua risteytykseen sen mukaan, kuinka hyvä vaihtoehto yksittäinen yksilö on suhteessa muihin yksilöihin sopivuusfunktion mukaan (Reeves, 2010). Jokaiselle yksilölle tulee laskea arvoalue, jonka suuruus määräytyy sen mukaan, kuinka suuren osuuden yksilön sopivuusfunktion tulos kattaa kaikkien yksilöiden sopivuusfunktion tulosten summasta. Esimerkiksi jos yksilön arvoalue on 0.12–0.42 on kyseisen yksilön sopivuusfunktion tuloksen suhteellinen osuus kaikkien yksilöiden sopivuusfunktion tuloksien summasta 30 %. Koodiesimerkki 4 havainnollistaa rulettivalinnan suoritusta.

```
}
   laske jokaiselle yksilölle arvoalue yksilön sopivuusfunktion tuloksen
   suhteellisen osuuden mukaan (esim. 0.00–0.32)
   WHILE (populaation verran yksilöitä ei ole valittu) {
      generoi satunnaisluku väliltä 0–1
      säilö se yksilö, jonka arvoalueelle luku osui
   }
}
```

Koodiesimerkki 4. Rulettivalinta.

Koska rulettivalinnassa yksilöt valitaan risteytykseen vuorotellen ja perustuen todennäköisyyteen, on mahdollista, että paras vaihtoehto ei tule valituksi (Hancock, 2002). Rulettivalinnan ongelmia voidaan vähentää käyttämällä *stokastista yleistä näytteenottoa* (Stochastic Universal Sampling, SUS), jossa vanhemmat valitaan yhdellä kertaa tasaisin välimatkoin. Kuva 1 havainnollistaa kumpaakin esitettyä valintaa. Kuvassa 1 nuoli esittää lukua, johon valinta on sillä kertaa osunut ja luvut 1–5 esittävät viittä eri vaihtoehtoa (yksilöä). Jokaiselle vaihtoehdolle on annettu alue sen mukaan, kuinka hyvä kyseinen vaihtoehto on suhteessa muihin vaihtoehtoihin. Kuvassa 1 vaihtoehto 1 on paras, koska se on saanut suurimman alueen, eli välin 0.00–0.32. Kuvasta 1 näemme, että oikealla puolella oleva SUS-valinta on yhdellä kerralla onnistunut valitsemaan 5 vanhempaa kun taas vasemmalla oleva rulettivalinta on valinnut vain yhden ja tarvitsee vielä lisää valintoja, jotta tarvittava määrä vanhempia risteytykseen saadaan valittua. Kuvasta 1 voidaan myös huomata, että SUS-valinnassa yksi nuolista osuu aina väistämättä sopivuusfunktion mukaan parhaimpaan yksilöön (eli yksilöön, jolla on suurin mahdollisuus tulla valituksi).



Kuva 1. Rulettivalinta (vasemmalla) ja SUS-valinta (oikealla) (Reeves, 2010).

Jotta ruletti- tai SUS-valintaa voidaan käyttää, on tiedettävä yksilön vahvuus suhteessa koko populaatioon. Tätä taustaoletusta on suurilla populaatioilla kuitenkin joskus vaikeaa

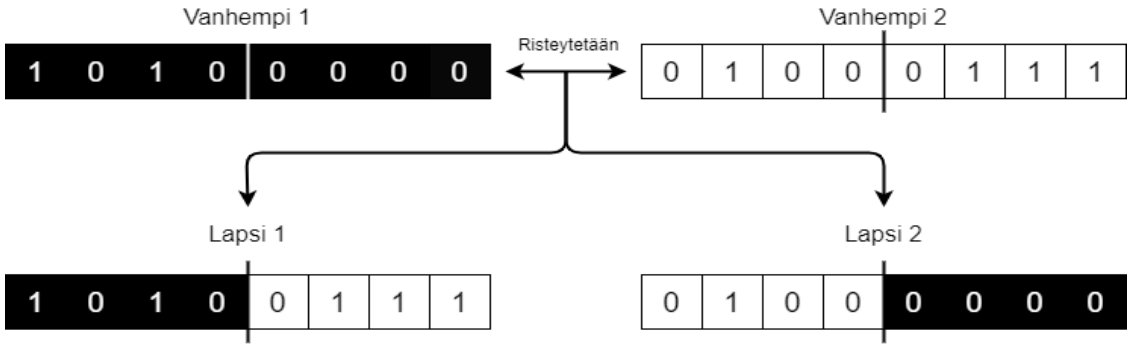
tai jopa mahdotonta saavuttaa. Tällöin *turnausvalinnan* (tournament selection) käyttäminen voi olla oikea ratkaisu. Turnausvalinnassa valitaan joukko ratkaisuja ja verrataan niitä toisiinsa, minkä jälkeen paras ratkaisu etenee vanhemmaksi risteytykseen (Reeves, 2010).

Monet valintamenetelmät kärsivät siitä, että paras vaihtoehto ei välttämättä tule aina valituksi, jolloin pitkänkin ajan jälkeen hankittu hyvä ratkaisu menetetään. Tähän ratkaisuna toimii *elitismi* (elitism), jossa tietty määrä parhaimpia ratkaisuja saa jatkaa risteytykseen ilman tarvetta tulla valituksi valintavaiheessa (Mitchell, 1999).

2.4 Risteytys

Geneettisen algoritmin yhteydessä ja evoluutio-ohjelmoinnissa yleensäkin, *risteytyksellä* (crossover) tarkoitetaan kahden yksilön piirteiden yhdistämistä yhdeksi jälkeläiseksi (Eiben ja Smith, 2015).

Risteytyksessä *yhden kohdan* (one-point) risteytyksellä tarkoitetaan jälkeläisten muodostamista niin, että vanhemmat yhdistetään yhdestä kohtaa (Man et al., 1999). Esimerkiksi jos yksilöt A = 1001 ja B = 0010 yhdistetään puolesta välistä, saadaan yksilöt 1010 ja 0001. Yhden kohdan risteytyksessä on kuitenkin useita eri ongelmia. Esimerkiksi yksilöä 1011 on mahdotonta saada yksilöistä A ja B edellä mainitulla tavalla. Eshelman ja muut (1989) nostavat ongelmaksi myös *paikkakohtaisen vääristymän* (positional bias), joka tarkoittaa, että uuden jälkeläisen genotyypin rakenne on riippuvainen siitä, missä kohtaa tietyt alleelit ovat sen vanhempien genotyyppien rakenteessa. Kuva 2 havainnollistaa risteytystä, jonka leikkauskohta jakaa genotyypit symmetrisesti kahteen osaan.



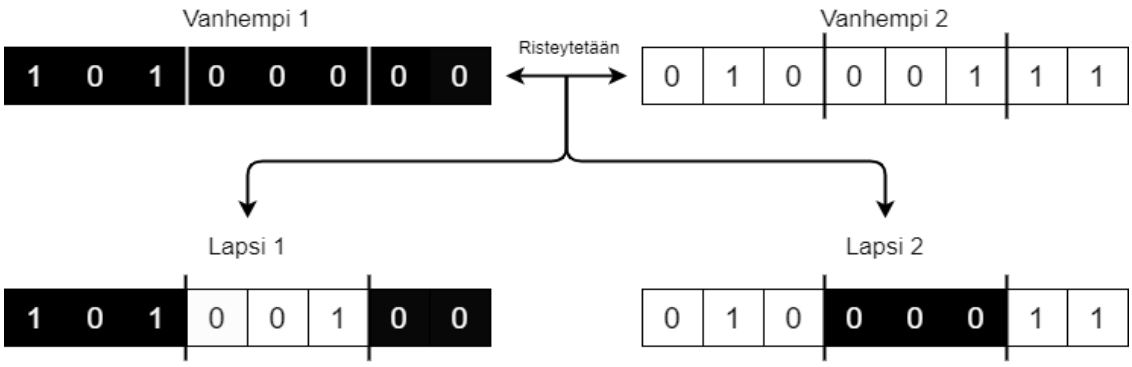
Kuva 2. Yhden kohdan risteytys.

Koodiesimerkki 5 esittelee yhden kohdan risteytyksen etenemisen.

```
{
   WHILE (kaikkia valittuja ei ole risteytetty) {
      ota 2 yksilöä risteytettäväksi
      risteytä valitut yksilöt (esimerkiksi puolestavälistä)
      säilö uudet yksilöt
   }
   korvaa vanha populaatio uusilla yksilöillä
}
```

Koodiesimerkki 5. Yhden kohdan risteytys.

On myös mahdollista suorittaa *m-kohdan* (m-point) risteytys, jolloin risteytyksessä valitaan jokin m-määrä kohtia risteytettäväksi. Esimerkiksi suorittamalla 2-kohdan risteytys leikkaamalla edellisessä esimerkissä esiteltyjen yksilöiden A:n ja B:n lokuksen 3 kummaltakin puolelta, saadaan jälkeläiset 1011 ja 0000. Tässä taas ongelmaksi muodostuu ns. *jakauman vääristymä* (distributional bias), joka aiheutuu siitä, kun yksilöiden rakenteet muuttuvat paljon monen kohdan risteytyksen takia (Eshelman et al., 1989). Kuva 3 havainnollistaa 2-kohdan risteytystä.



Kuva 3. 2-kohdan risteytys.

Tästä on siis ymmärrettävä, että jakauman vääristymää on mahdollista vähentää vähentämällä kohtia, jotka valitaan risteytykseen, mutta samalla paikkakohtainen vääristymä kasvaa ja toisinpäin. Eshelmanin ja muiden (1989) tutkimusten mukaan optimaalinen määrä kohtia valittavaksi on 8.

2.5 Mutaatio

Mutaatiossa (mutation) tarkoituksena on muokata sattumanvaraisesti valittujen yksilöiden alleeleita (Reeves, 2010). Esimerkiksi binäärimerkkijono 0010 voisi olla mutaation jälkeen 1010. Mutaation tarkoitus on siis sisällyttää sattumanvaraisuutta läpi algoritmin

suorittamisen, mutta sen ei kannata olla kuitenkaan liian todennäköinen, koska mitä enemmän mutaatiota esiintyy, sitä enemmän algoritmin suoritus perustuu satunnaisuuteen. Esimerkiksi Laabadi ja muut (2018) käyttävät 1–0.1 prosentin mutaatioastetta soveltaessaan geneettistä algoritmia.

Mutaation tarkoitus on myös estää monimuotoisuuden katoaminen algoritmin suorituksen yhteydessä (Reeves, 2010). Ilman mutaatiota on mahdollista, ettei parhaaseen ratkaisuun voida päästä nykyisestä populaatiosta vaan algoritmi ikään kuin juuttuu ratkaisujen joukkoon, joka on kaikkien ratkaisujen osajoukko. Tällöin on mahdollista löytää pelkästään *lokaali optimi* (local optimum), eli tämän osajoukon paras ratkaisu (Eiben ja Smith, 2015). Mutaation avulla on mahdollista paeta tästä osajoukosta ja löytää *globaali optimi* (global optimum), eli paras ratkaisu kaikista vaihtoehdoista (Reeves, 2010).

Ajatellaan, että ratkaisujen joukko koostuu binäärimerkkijonoista 1000 ja 0001. Pelkästään risteyttämällä näitä kahta genotyyppiä ei ole mahdollista saada esimerkiksi binäärimerkkijonoa 0010, sillä kumpikaan merkkijono ei sisällä alleelia 1 lokuksessa 3. Mutaatiolla kuitenkin edellä mainitun binäärimerkkijonon saavuttaminen on mahdollista, koska mutaation stokastisen luonteen takia seuraavan sukupolven yksilöiden ei tarvitse olla täysin omien vanhempiensa kombinaatioita.

Vaikka risteytystä ja mutaatiota käytetäänkin usein geneettisen algoritmin suorittamisessa yhdessä, tämän päätöksen tekeminen riippuu aina käsillä olevasta ongelmasta. Kuten Reeves (2010) huomauttaa, on mahdollista, että geneettisessä algoritmissa käytetään pelkästään joko risteytystä tai mutaatiota.

2.6 Algoritmin lopetus

Geneettisen algoritmin luonteen takia, on mahdollista, että algoritmin suoritus jatkuu loputtomiin. Tämän takia geneettisen algoritmin lopetuksen määrittäminen on tärkeää, ja sen voi suorittaa monella eri tavalla.

Mikäli paras ratkaisu on tiedossa, on lopetusehdon luominen hyvin yksinkertaista, koska tällöin algoritmin voi lopettaa, kun päästään parhaaseen ratkaisuun tai tarpeeksi lähelle sitä. Näin ei kuitenkaan usein ole.

Reeves (2010) ehdottaa vaihtoehdoiksi esimerkiksi maksimimäärän asettamista sukupolville, algoritmin ajamista tietyn ajan verran tai populaation monimuotoisuuden tarkkailua ja algoritmin lopetusta, mikäli monimuotoisuus laskee tietyn rajan alle. Lopetusehto on siis mahdollista asettaa monella eri tavalla, minkä takia algoritmin käyttäjällä on vastuu valita tilanteeseen sopiva lopetusehto optimaalisten tulosten saavuttamiseksi.

3 Sovelluksia

Geneettiselle algoritmille on vuosien saatossa löydetty monia eri sovelluksia. Tässä työssä keskitytään kahteen yleiseen ongelmaan, joita on mahdollista soveltaa moniin tosielämän ongelmiin. Kummatkin työssä esitellyt ongelmat ovat hyvin laajoja, joten tarkoituksena ei ole tarjota syvällistä katsausta ongelmiin, vaan ainoastaan tarkastella ongelmia geneettisen algoritmin näkökulmasta ja perehtyä geneettisen algoritmin soveltamiseen uusimpien tutkimusten pohjalta.

3.1 Kauppamatkustajan ongelma

Kauppamatkustajan ongelma (Travelling Salesman Problem, TSP) on hyvin yleinen verkkoteorian ongelma, jota on mahdollista soveltaa useaan ongelmaan niin tieteen kuin tekniikan aloilla (Agrawal ja Jain, 2020). Koska kauppamatkustajan ongelmaa voidaan soveltaa hyvin monessa tosielämän ongelmassa, ja koska geneettinen algoritmi sopii hyvin kyseisen ongelman ratkaisuun, näen hyödylliseksi sisällyttää tämän ongelman esittelemisen geneettisen algoritmin näkökulmasta tähän työhön. Esimerkiksi elintarvikkeita usean eri kaupan välillä kuljettavan rekan lyhimmän tai nopeimman reitin määrittely voidaan ratkaista soveltamalla kauppamatkustajan ongelman ratkaisua.

Kauppamatkustajan ongelmassa on tarkoitus käydä jokaisessa ennalta määrätyssä pisteessä palaten aloituspisteeseen lyhintä reittiä (Agrawal ja Jain, 2020). Tämä ongelma voidaan ratkaista laskemalla jokaisen reitin pituus pisteiden välillä, mutta ratkaisuun pääseminen tällä tavalla kestää usein liian pitkään. On siis kannattavaa käyttää jotain muuta algoritmia ongelman ratkaisemiseen. Yksi näistä on geneettinen algoritmi, jonka avulla kauppamatkustajan ongelma on mahdollista ratkaista usealla eri tavalla.

Agrawal ja Jain (2020) ehdottavat geneettisen algoritmin käyttämistä niin, että puolet alkupopulaatiosta muodostetaan satunnaisesti valituista genotyypeistä (reiteistä) ja loput taas niin, että valitaan aina lähin seuraava piste, välttäen kuitenkin jo valittuja reittejä. Muuten Agrawal ja Jain (2020) käyttävät jo läpikäytyä yksinkertaisen geneettisen algoritmin rakennetta. Kauppamatkustajan ongelman kaltaisten graafiongelmien kanssa on tärkeää määritellä risteytys ongelman luonteen mukaan (Sakai et al., 2020), sillä jo esitellyn yksinkertaisen geneettisen algoritmin risteytyksen soveltaminen sellaisenaan ei sovi tähän ongelmaan. Tämä sen takia, koska perinteinen risteytys (esim. yhden kohdan risteytys) ei takaa sitä, että jokainen alleeli tulee esiintymään vain kerran risteytetyn lapsen genotyypissä. Seuraavaksi esittelenkin muutaman eri tavan toteuttaa risteytys kauppamatkustajan ongelman ratkaisemiseksi. Muita geneettisen algoritmin operaatioita ei

käydä sen tarkemmin tämän ongelman kontekstissa läpi, sillä niiden soveltaminen on mahdollista suorittaa triviaalisti jo luvussa 2 esiteltyjä geneettisen algoritmin vaiheita soveltamalla. Esitän seuraavissa esimerkeissä kauppamatkustajan reitin tapaan (1 2 3 4 5 6 7 8 9), jossa jokainen numero edustaa eri kaupunkia.

3.1.1 Osittain kuvattu risteysoperaattori

Yksinkertainen tapa toteuttaa risteytys tämän ongelman ratkaisemiseksi on käyttää *osittain kuvattua risteytysoperaattoria* (Partially Mapped Crossover Operator, PMX). PMX:ssä valitaan ensiksi satunnaisesti leikkauskohdat, minkä jälkeen leikkauskohtien välissä oleva pätkä geenejä asetetaan toiseen vanhempaan (Hussain et al., 2017). Genotyypit 1 esittää esimerkissä käytettävät genotyypit. Merkillä ”|” tarkoitetaan leikkauskohtaa. Genotyypin nimeämisissä käytetään kirjainta P indikoimaan, että kyseessä on vanhempi, kun taas C indikoi risteytyksessä muodostettua lasta.

P1 = (1 4 2 | 3 5 6 | 8 9 7)
P2 = (3 8 2 | 9 1 6 | 7 4 5)
Genotyypit 1.

Genotyypit 2 esittää miltä genotyyppi näyttää, kun leikkauskohtien välinen pätkä on vaihdettu. Leikkauskohtien välissä vaihdettiin luvut 3 ←→ 9, 5 ←→ 1 ja 6 ←→ 6.

C1 = (X X X | 9 1 6 | X X X)
C2 = (X X X | 3 5 6 | X X X)
Genotyypit 2.

Genotyypit 3 kohdassa algoritmi on pyrkinyt asettamaan leikkauskohtien ulkopuolella olevat pätkät takaisin, mutta koska osa numeroista on jo valittu edellisessä vaiheessa eikä samoja numeroita voi valita kahteen kertaan (sillä tarkoitus on käydä jokaisessa pisteessä vain kerran), osa numeroista täytyy selvittää erikseen.

C1 = (X 4 2 | 9 1 6 | 8 X 7)
C2 = (X 8 2 | 3 5 6 | 7 4 X)
Genotyypit 3.

Puuttuvat numerot selvitetään niin, että katsotaan minkä luvun kanssa jo valittu luku vaihdettiin ja mikäli sitä ei ole vielä valittu, asetetaan se puuttuvan luvun paikalle. Esimerkiksi genotyypin P1 ensimmäisen puuttuvan luvun paikalla oli aluksi luku 1 ja tämä luku vaihtoi alussa paikkaa luvun 5 kanssa, joten asetetaan luku 5 genotyypin C1 ensimmäiseen

lokukseen. Käydään loput puuttuvat luvut samalla logiikalla läpi ja saadaan Genotyypit 4 esimerkin mukaiset rakenteet.

C1 = (5 4 2 | 9 1 6 | 8 3 7)
C2 = (9 8 2 | 3 5 6 | 7 4 1)
Genotyypit 4.

3.1.2 OX operaattori

Toinen tapa toteuttaa risteytys on *OX operaattori* (OX operator). Moscato (1989) esittelee artikkelissaan OX operaattorin Goldbergin (1989) kirjan pohjalta. Käytetään tässä työssä samaa tapaa, sillä se on intuitiivinen ja helppo ymmärtää. Kuten Moscato (1989) selittää viitatessaan Goldbergin (1989) kirjaan, tässä risteytyksessä valitaan satunnaisesti yksi pätkä genotyypistä, jonka toinen vanhempi luovuttaa toiselle. Genotyypit 5 esittää alkutilanteen vanhemmat, jossa ”|” -merkkien välinen pätkä luovutetaan G1 genotyypistä genotyyppiin G2.

G1 = (7 9 1 | 6 5 4 | 3 8 2)
G2 = (5 7 6 | 1 3 2 | 9 8 4)
Genotyypit 5.

Jotta voidaan lahjoittaa pätkä genotyypistä G1 genotyyppiin G2, täytyy aluksi poistaa G2 genotyypistä ne alleelit, jotka löytyvät lahjoitetusta pätkästä. Genotyypit 6 havainnollistaa tätä poistoa.

G1 = (7 9 1 | 6 5 4 | 3 8 2)
G2 = (X 7 X | 1 3 2 | 9 8 X)
Genotyypit 6.

Seuraavaksi siirretään oikeanpuoleisesta leikkauskohdasta kaikkia numeroita vasemmalle, jotta saadaan tilaa luovutetulle pätkälle. Genotyypit 7 havainnollistaa tilannetta, kun tämä siirtyminen on tehty.

G1 = (7 9 1 | 6 5 4 | 3 8 2)
G2 = (1 3 2 | X X X | 9 8 7)
Genotyypit 7.

Siirtymisen jälkeen asetetaan luovutettava pätkä genotyyppiin G2. Genotyypit 8 esittää lopullisen version luovutuksen jälkeen.

G1 = (7 9 1 | 6 5 4 | 3 8 2)
G2 = (1 3 2 | 6 5 4 | 9 8 7)
Genotyypit 8.

Tehdään sama operaatio, mutta toisin päin käyttäen alkuperäistä (Genotyypit 5) G2 genotyyppiä pohjana genotyypin G1 muuttamiselle. Tällöin jo edellä esitellyllä logiikalla saadaan Genotyypit 9 esimerkin tulos.

G1 = (6 5 4 | 1 3 2 | 8 7 9)
G2 = (1 3 2 | 6 5 4 | 9 8 7)
Genotyypit 9.

3.1.3 Muunnettu sykliristeytys

Hussain ja muut (2017) ehdottavat risteytyksessä käytettäväksi omaa muunnelmaansa *sykliristeytyksestä* (CX), jolle he antoivat nimeksi CX2, ja jonka he huomasivat toimivan testeissään paremmin kuin kaksi edeltävää risteytystä TSP:n ratkaisemiseksi. Tässä työssä ei käydä läpi sykliristeytystä erikseen vaan keskitytään pelkästään Hussainin ja muiden (2017) kehittämään muunnettuun versioon. Genotyypit 10 esittelee esimerkin genotyypit.

P1 = (1 2 3 4 5 6 7 8)
P2 = (2 7 5 8 4 1 6 3)
Genotyypit 10.

Ensiksi valitaan genotyypistä P2 ensimmäisen lokuksen geeni ja asetetaan se uuden genotyypin C1 vastaavaan lokukseen Genotyypit 11 esimerkin mukaan.

C1 = (2 X X X X X X X)
C2 = (X X X X X X X X)
Genotyypit 11.

Etsitään nyt alkuperäisen esimerkin (Genotyypit 10) P1 genotyypistä alleeli 2 ja nähdään, että genotyypin P2 vastaavassa lokuksessa on luku 7. Ei valita vielä tätä lukua 7 vaan käytetään samaa logiikkaa uudestaan ja etsitään genotyypistä P1 luvun 7 lokus ja asetetaan genotyypin P2 vastaavan lokuksen alleeli eli luku 6 uuteen lapseen C2. Esimerkki Genotyypit 12 havainnollistaa muutosta.

C1 = (2 X X X X X X X)
C2 = (6 X X X X X X X)
Genotyypit 12.

Nyt etsitään alkuperäisen esimerkin genotyypistä P2 se lokus, josta viimeisin löydetty luku (eli luku 6) löytyy genotyypistä P1. Tästä lokuksesta löydetään luku 1 ja asetetaan se lapseen C1. Etsitään lapsen C2 seuraava luku jo edellä mainitulla tavalla eli etsitään viimeisin luku (luku 1) genotyypistä P1 ja etsitään vastaavassa lokuksessa oleva luku genotyypistä P2, jolloin saadaan luku 2. Taaskaan ei vielä valita tätä lukua 2 vaan etsitään se genotyypistä P1 ja valitaan samassa lokuksessa oleva luku genotyypistä P2. Saadaan luku 7 ja asetetaan se lapseen C2. Edellä mainittua logiikkaa jatketaan joko kunnes molemmat lapset ovat täysiä tai päästään Genotyypit 13 havainnollistamaan tilanteeseen.

C1 = (2 1 6 7 X X X X)
C2 = (6 7 2 1 X X X X)
Genotyypit 13.

Esimerkissä Genotyypit 13 on tapahtunut konflikti. Nyt ei voida valita lukua, joka on genotyypin P2 samassa lokuksessa kuin missä genotyypin P1 luku 1 (eli viimeisin löydetty luku) sijaitsee, sillä tämä luku on 2 ja se on jo valittu kyseiseen lapseen. Tässä tilanteessa muokataan vanhempia niin, että poistetaan jo valitut numerot, jolloin vanhemmat näyttävät esimerkin Genotyypit 14 kaltaisilta.

P1 = (3 4 5 8)
P2 = (5 8 4 3)
Genotyypit 14.

Nyt aloitetaan jo edellä mainittu logiikka uudelleen alusta ja jatketaan risteytystä, kunnes lopulta päästään Genotyypit 15 esimerkin tilanteeseen, jossa lapset on muodostettu kokonaisuudessaan ja risteytys on saatettu päätökseen.

C1 = (2 1 6 7 5 3 8 4)
C2 = (6 7 2 1 8 4 5 3)
Genotyypit 15.

3.2 Reppuongelma

Seuraavaksi käsittelen geneettisen algoritmin soveltamista *reppuongelmaan* (Knapsack problem), joka on hyvin yleinen kombinatorinen optimointiongelma (Laabadi et al., 2018). Reppuongelmassa jokaisella yksilöllä on paino ja arvo ja valitut yksilöt laitetaan reppuun, jolla on jokin maksimipaino. Tarkoituksena on valita se kombinaatio yksilöitä, joilla yksilöiden yhteenlaskettu arvo on suurin, mutta joiden yhteenlaskettu paino on vähemmän tai yhtä paljon kuin repun maksimipaino (Pradhan et al., 2014).

Tässä kappaleessa esitellään reppuongelman ratkaiseminen jo aikaisemmin käydyn yksinkertaisen (klassisen) geneettisen algoritmin avulla, minkä jälkeen esitellään tapoja muokata geneettistä algoritmia, jotta reppuongelma on mahdollista ratkaista nopeammin ja paremmilla tuloksilla. Genotyypit esitetään tapaan {0 1 0 0 1 1}, jossa 0 tarkoittaa ettei yksilöä valittu reppuun, kun taas 1 tarkoittaa että yksilö valittiin reppuun.

3.2.1 Geneettisen algoritmin soveltaminen reppuongelmaan

Aloitetaan ongelman tarkastelu esittelemällä 4 eri yksilöä C_1, C_2, C_3 ja C_4, joiden painot (W) ja arvot (P) nähdään taulukosta 1. Repun maksimipainoksi valitaan 20.

Taulukko 1. Yksilöiden attribuutit.

Yksilö	W	P
C_1	10	4
C_2	12	8
C_3	8	2
C_4	6	3

Tarkoituksena on maksimoida kaava

$$\max = 4C_1 + 8C_2 + 2C_3 + 3C_4$$

ehdolla

$$10C_1 + 12C_2 + 8C_3 + 6C_4 \leq 30$$

Kaava 2. Ehto.

Alustetaan populaatio satunnaisesti luomalla 4 eri genotyyppiä, jotka täyttävät kaavan 2 ehdon. Populaation genotyypit esitellään taulukossa 2.

Taulukko 2. Populaation alustus.

Genotyyppi	P
1001	7
0110	10
1010	6
0101	11

Valinta voidaan suorittaa jo aikaisemmin esitellyllä rulettivalinnalla. Hakimi ja muut (2016) vertailivat tutkimuksessaan useaa eri risteytysmetodia reppuongelman ratkaisuun, joista parhaimmaksi nousi jo aikaisemminkin esitelty 2-paikan risteytys. Käytetään siis 2-paikan risteytystä ja risteytetään valitut genotyypit. Suoritetaan 2-paikan risteytys seuraavaan tapaan (”|” = leikkauskohta): {0 | 1 0 | 1}. Rulettivalinnan ja 2-paikan risteytyksen jälkeen muodostettu uusi populaatio esitetään taulukossa 3.

Taulukko 3. Uusi populaatio.

Genotyyppi	P
0111	13
0100	8
1110	14
0010	2

Nyt on mahdollista suorittaa mutaatio, jonka jälkeen algoritmin suoritus jatkuu, kunnes lopetusehto täytetään. Lopetusehdoksi voidaan valita esimerkiksi tietty määrä sukupolvia (Laabadi et al., 2018).

3.2.2 Geneettisen algoritmin parannuksia reppuongelmaan

Edellä esiteltyä geneettisen algoritmin sovellusta voidaan jatkaa tai muokata, jotta geneettinen algoritmi suoriutuisi paremmin reppuongelman ratkaisemisessa. Seuraavaksi esittelen pintapuolisesti kaksi tapaa muokata geneettistä algoritmia reppuongelman ratkaisemiseksi tuoreiden tutkimusten pohjalta. Syvällisemmän ymmärryksen kustakin tavasta voi saada tutustumalla lähdekirjallisuuteen.

Pradhan ja muut (2014) suosittelevat lisäävän perinteiseen geneettiseen algoritmiin yhden lisävaiheen suoritettavaksi mutaation jälkeen. Tämä lisävaihe, nimeltään *attribuuttien vähentämisen tekniikka* (Attribute Reduction Technique), käyttää hyväkseen *suurpiirteistä joukko-oppia* (Rough Set Theory) löytääkseen parhaimmat yksilöt kyseisestä ongelmasta. Tämän takia attribuuttien vähentämisen tekniikka löytää aina optimiratkaisun, kun taas yksinkertaisen geneettisen algoritmin tapauksessa näin ei aina ole. (Pradhan et al., 2014)

Montazeri ja muut (2017) esittelee tutkimuksessaan geneettisen algoritmin muunnelman, joka pohjautuu *saarimaiseen geneettiseen algoritmiin* (Island Genetic Algorithm, IGA), jossa ideana on suorittaa geneettinen algoritmi monella eri populaatiolla rinnakkain. Montazeri ja muut (2017) muokkaavat tätä ideaa reppuongelmaan niin, että he aluksi etsivät monta eri lokaalia optimia ajamalla monta geneettistä algoritmia rinnakkain. Tämän jälkeen he valitsevat jokaisen lokaalin optimin ja asettavat ne yhteen pääalgoritmiin, jolla he pyrkivät löytää globaalin optimin.

4 Yhteenveto

Tutkielmassa käsiteltiin geneettistä algoritmia, sekä perehdyttiin kahteen ongelmaan, johon geneettistä algoritmia voidaan soveltaa. Geneettisen algoritmin tavoitteena on löytää ratkaisu käsillä olevaan ongelmaan evoluutiobiologiasta tuttujen prosessien avulla. Geneettistä algoritmia esitellessä käytiin läpi vaiheet, jotka määrittelevät geneettisen algoritmin etenemisen: populaation alustuksen, sopivuusfunktion, valinnan, risteytyksen, mutaation ja algoritmin lopetuksen. Lopuksi tarkasteltiin geneettisen algoritmin soveltamista kauppamatkustajan ongelmaan ja reppuongelmaan.

Populaation alustuksessa tarkasteltiin tapoja populaation koon määrittämiseen, ja siihen miten populaatio tulisi alustaa. Tämän jälkeen tarkasteltiin sopivuusfunktiota ja sen roolia geneettisen algoritmin suorituksessa, minkä jälkeen syvennyttiin valintaan. Valinnan yhteydessä esiteltiin ruletti- ja SUS-valinta, sekä elitismin rooli. Valinnan jälkeen määriteltiin risteytys, sekä käytiin läpi kaksi tapaa toteuttaa risteytys: yhden kohdan risteytys ja m-kohdan risteytys. Seuraavaksi syvennyttiin mutaatioon ja sen rooliin geneettisessä algoritmissa, ja lopuksi lukijalle esiteltiin erilaisia keinoja algoritmin lopettamiseen.

Luvussa 3 käytiin läpi geneettisen algoritmin soveltamista ja esiteltiin konkreettiset esimerkit ongelmista, joihin geneettinen algoritmi soveltuu. Kauppamatkustajan ongelmassa esiteltiin, kuinka geneettistä algoritmia voidaan soveltaa verkkoteorian ongelmiin.

Koska kauppamatkustajan ongelmassa muut geneettisen algoritmin osa-alueet ovat helposti sovellettavissa jo aiemmin luvussa 2 esitellystä yksinkertaisesta geneettisestä algoritmista, keskityimme pääasiallisesti pelkästään eri tapoihin toteuttaa risteytys kyseiseen ongelmaan soveltuvasti. Reppuongelmassa käytiin aluksi läpi geneettisen algoritmin soveltamista kombinatoriseen optimointiongelmaan. Tämän jälkeen esiteltiin nykypäivän tutkimusten avulla muutamia tapoja muokata geneettistä algoritmia, jotta algoritmista saadaan tehokkaampi.

Tulevaisuuden tutkimusten kannalta mielenkiintoisia kysymyksiä voisi löytyä esimerkiksi neuroverkkojen ja geneettisen algoritmin välisistä yhteyksistä. Vaikkakin neuroverkot ja geneettiset algoritmit usein pyrkivät ratkaisemaan erityyppisiä ongelmia, voidaan geneettistä algoritmia hyödyntää neuroverkkojen tukena esimerkiksi neuroverkon kouluttamisessa. Myös tässä tutkimuksessa nopeasti läpi käytyihin ongelmiin kuten geneettisen algoritmin ennenaikaiseen konvergoitumiseen epäoptimaaliseen ratkaisuun voitaisiin syventyä tarkemmin ja löytää mielenkiintoisia tutkimuskysymyksiä.

Lähdeluettelo

Agrawal, M. & Jain, V. (2020). Applying Improved Genetic Algorithm to Solve Travelling Salesman Problem. *Second International Conference on Inventive Research in Computing Applications* (ss. 1194-1197). doi: 10.1109/ICIRCA48905.2020.9182884

Eiben, A. E. & Smith, J. E. (2003). *Introduction to Evolutionary Computing.* (1st ed.). Springer. doi: https://doi.org/10.1007/978-3-662-05094-1

Eiben, A. E. & Smith, J. E. (2015). *Introduction to Evolutionary Computing.* (2nd ed.). Springer. doi: https://doi.org/10.1007/978-3-662-44874-8

Eshelman, L. J., Caruana, R. & Schaffer, J. D. (1989). Biases in the Crossover Landscape. *Proceedings of the 3rd International Conference on Genetic Algorithms* (ss. 10-19). Fairfax, Virginia: Morgan Kaufmann Publishers.

Goldberg, D. E. (1989). *Genetic Algorithms in Search Optimisation and Machine Learning*. Addison-Wesley.

Hakimi, D., Oyewola, D., Yahaya, Y. & Bolarin, G. (2016). Comparative analysis of genetic crossover operators in knapsack problem. *Journal of Applied Sciences and Environmental Management,* 20(3), (ss. 593-596). doi: 10.4314/jasem.v20i3.13

Hancock, P. J. B. (2002). *An Empirical Comparison of Selection Methods in Evolutionary Algorithms*. Selected Papers from AISB Workshop on Evolutionary Computing. doi: 10.1007/3-540-58483-8_7

Hussain, A., Muhammad, Y. S., Sajid, N. & Hussain, I. (2017). Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. *Computational Intelligence and Neuroscience,* 2017(1), (ss. 1-7). doi: 10.1155/2017/7430125

Koumousis, V. K. & Katsaras, C. P. (2006). A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation,* 10(1), (ss. 19-28). doi: 10.1109/TEVC.2005.860765

Laabadi, S., Naimi, M., El Amri, H. & Achchab, B. (2018). A hybrid genetic algorithm for solving 0/1 Knapsack Problem. *LOPAL '18: Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications* (ss. 1-6). doi: 10.1145/3230905.3230907

Man, K. F., Tang, K. S. & Kwong, S. (1999). *Genetic Algorithms.* (1st ed.). Springer. doi: https://doi.org/10.1007/978-1-4471-0577-0

Mitchell, M. (1999). *An Introduction to Genetic Algorithms*. (1st ed.) MIT Press.

Montazeri, M., Kiani, R. & Rastkhadiv, S. S. (2017). A new approach to the Restart Genetic Algorithm to solve zero-one knapsack problem. *IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)* (ss. 0050-0053). doi: 10.1109/KBEI.2017.8324863

Moscato, P. (1989). On Genetic Crossover Operators for Relative Order Preservation. *C3P-Report.*

Pradhan, T., Israni, A. & Sharma, M. (2014). Solving the 0–1 Knapsack problem using Genetic Algorithm and Rough Set Theory. *IEEE International Conference on Advanced Communications, Control and Computing Technologies* (ss. 1120-1125). doi: 10.1109/ICACCCT.2014.7019272

Reeves, C. R. (2010). Genetic Algorithms. In M. Gendreau & J. Y. Potvin (Ed.), *Handbook of Metaheuristics* (ss. 109-139). New York City, NY: Springer International Publishing

Sakai, M., Hanada, Y. & Orito, Y. (2020). Edge Assembly Crossover with Tabu for Traveling Salesman Problem. *IEEE Congress on Evolutionary Computation (CEC)* (ss. 1-6). Glasgow, UK doi: 10.1109/CEC48606.2020.9185837